## Macros

## Announcements

## Expressions

## Discussion Question: Pythagorean Theorem

Quick quasiquotation review: `` `(+ ,(* 2 3) 1) `` evaluates to `(+ 6 1)`

Add `` ` `` and `,` in some blanks so that the second expression evaluates to `(+ (* a a) (* b b))`

```
_(define (square-expr term) `( _* ,term ,term))

`( _+ ,( _square-expr `a) ,( _square-expr `b))
```

(Demo)

## Macros

## Macros Perform Code Transformations

A macro is an operation performed on the source code of a program before evaluation

Macros exist in many languages, but are easiest to define correctly in a language like Lisp

Scheme has a **define-macro** special form that defines a source code transformation

```
(define-macro (twice expr)
  (list 'begin expr expr))
```

```
> (twice (print 2))   ▶ (begin (print 2) (print 2))
2
2
```

macro can: duplicate before evaluate "expr"

Evaluation procedure of a macro call expression:

- Evaluate the operator sub-expression, which evaluates to a macro
- Call the macro procedure on the operand expressions *without evaluating them first*
- Evaluate the expression returned from the macro procedure

(Demo) (define x (print 2))

For Macro

## Discussion Question

Define a macro that evaluates an expression for each value in a sequence

```
scm> (map (lambda (x) (* x x)) (2 3 4 5))
(4 9 16 25)
```

```
(define-macro (for sym vals expr)
  (list 'map ____________(list 'lambda (list sym) expr) vals)____________)
```

```
scm> (for x (2 3 4 5) (* x x))
(4 9 16 25)
```

(Demo)

## Trace

## Tracing Recursive Calls

```python
def trace(fn):
    def traced(n):
        print(f'{fn.__name__}({n})')
        return fn(n)
    return traced

@trace
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n - 1)
```

```scheme
(define fact (lambda (n)
  (if (zero? n) 1 (* n (fact (- n 1))))))

(define original fact)
(define fact (lambda (n)
           (print (list 'fact n))
           (original n)))
```

```
>>> fact(5)
fact(5)
fact(4)
fact(3)
fact(2)
fact(1)
fact(0)
120
```

```
scm> (fact 5)
(fact 5)
(fact 4)
(fact 3)
(fact 2)
(fact 1)
(fact 0)
120
```

(Demo)